Planning with Incrementally Created Graphs

Stephen Balakirsky[1] Otthein Herzog[2]

Abstract

A framework for planning algorithms that is both optimal and complete will be presented. The algorithm allows for the planning of optimal paths through a multi-level hierarchy of annotated graph spaces. A rich world model that contains multi-resolution attributes, costs, and predicates controls the integrated incremental construction and evaluation of the planning graph in dynamic environments. The properties of this framework are proven and a number of example problem domains are presented that show how this framework can incorporate both hard and soft constraints during both graph construction and arc evaluation.

Keywords

Planning graph; world model; optimal plan; robotic planning

Introduction

Many of today's challenging planning problems may be solved through the use of discrete representations. These include such diverse problems as robotic vehicle planning, parts assembly planning, and domain independent planning systems. These problems may also include dynamic environments in which real time replanning is required and in which novel planning and multi-resolution data requirements exist. The problem space for these tasks may be flat (single level) or may involve multilevel hierarchies.

Once the problem has been mapped to a discrete form, a planning graph may be created and evaluated in order to plan sequences of actions that allow the system to meet specific goals. In traditional graph planning techniques, this planning graph is fully constructed beforehand and stored for use during the planning cycle. In this paper, a new framework is presented that incorporates and extends existing graph planning and search techniques and that allows for the incremental construction and evaluation of the planning graph during the execution of the planning cycle. A rich world model that may contain data at multiple resolutions and in multiple formats controls this incremental construction and evaluation.

It will be shown that this framework allows for significant savings in both time and memory requirement for the planning system. In addition, the plans created will be able to take into account both constraints on the possible state space and the cost/benefit of state transitions. Constraints on the planning space strictly prohibit certain state transitions, while cost/benefit analysis allows the system to prefer certain classes of state transitions to others.

[1] NIST, stephen@nist.gov

[2] Work performed while guest researcher at NIST from Center for Computing Technologies, University of Bremen, Germany, herzog@tzi.de

In addition to finding a set of state transitions that allow the system to transition from the current state to the goal state, later sections will show that this framework finds plans that are the optimal set of transitions (optimal path) necessary to traverse from the system's current state to the goal state with respect to a dynamic cost function and system constraints. In this case, optimality refers to the property that given a set of discrete states and possible state transitions, no lower cost walk of state transitions exists than the solution provided by this system for a given cost function. The properties of this framework are proven and a number of example problem domains are presented that show how this framework can incorporate hard and soft constraints during both graph construction and arc evaluation.

Framework Definitions

It is desired to define a planning space **S** that will allow for a cost optimal sequence of transitions to be enumerated for the task of transitioning from one element of **S** to another. In order to sequence the members of **S**, an ordered sequence T of points (t_1, …,t_m) in time and the node space ND will now be defined. The space **S** over $\mathbb{N}^n$, is a basic structure such that $s_{\vec{k}} \in \mathbf{S}$ where $\vec{k} = (k_1,\ldots,k_n)$, $k_1,\ldots,k_n \in \mathbb{N}$.

Definition 1: The *node space* **ND** is defined by (**S**×T) and nd ∈ **ND** = $(s_{\vec{k}}, t_i)$, $t_i \in T$.

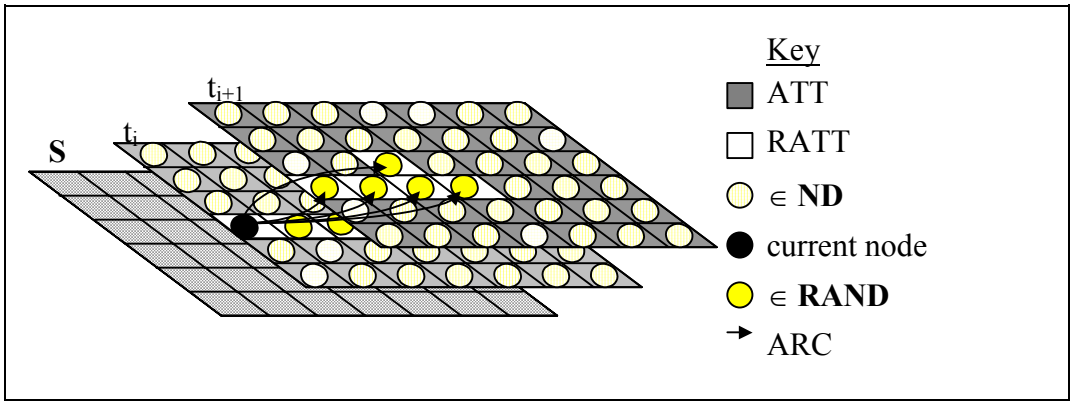

Figure 1: Depiction of spaces and sets from problem formulation.

The space **S** for n = 2 is depicted as a plane in Figure 1. The space **ND** is depicted as the partially filled spheres that lie in several time indexed planes.

Before one can discuss the optimality of a path, there must exist attributes with which to compute the cost/benefit of the path. Under the current construction of the node space, the only cost/benefit measure that can exist (and hence the only measure with which to compute optimality) is the number of node transitions that make up a plan. An additional objective of the planning system is to provide the infrastructure necessary to develop a

more complex notion of cost. In order to accomplish this, the notion of attributes will be introduced. These attributes capture all of the static and time varying features that can be associated with $s_{\vec{k}}$ at time t.

Definition 2: Define the set of *possible node general attributes* NGATT = {ngatt | ngatt is a possible feature of $nd_{\vec{k},t} \in$ **ND**}.

It is possible that a specific node $nd_{\vec{k},t}$ will not contain all of the possible general attributes. Therefore the set of actual node attributes NATT $\subseteq$ NGATT will be defined. ATT is defined as a set of sets, where each member $att_{\vec{k},t}$ is the set NATT of <u>all</u> features that can possibly be associated with $s_{\vec{k}}$ at time t.

Definition 3: Define the set of *actual node attributes* NATT = {natt | natt is a feature of $nd_{\vec{k},t} \in$ **ND**} and the set of *attribute set*s as ATT = $\{ NATT_{\vec{k},t} \mid \vec{k} = (k_1,\ldots,k_n), k_1,\ldots,k_n \in \mathbb{N}, t_i \in T \}$.

The set of attributes ATT is represented in Figure 1 as a collocated plane (the shaded regions) where each shaded rectangle represents a set of attributes NATT. To simplify future notation, an annotated node space **AND** may be defined that combines the node space **ND** and the set of attribute sets ATT.

Definition 4: An *annotated node space* **AND** is defined as **AND** = (**ND**×ATT) where $and_{\vec{k},t} \in$ **AND**.

If the planning space is feature rich, NATT may be quite large. Computing the cost/benefit ratio by utilizing all of the features in this potentially large set will needlessly increase the computational burden. Instead of utilizing NATT, a new set of node relevant features NRATT will be defined that only includes the features that are relevant for the current instantiation of the cost/benefit evaluation at a given time, and is a subset of NATT. RATT is then defined as the set of sets consisting of the feature sets NRATT and is a subset of ATT.

Definition 5: Define the set NRATT = $\{ nratt_{\vec{k},t_i} \mid nratt_{\vec{k},t_i} \subseteq natt_{\vec{k},t_i} \}$ to be the set of all domain dependent *node relevant attributes* for node $nd_{\vec{k},t_i}$ and the set of *relevant attribute set*s as RATT = $\{ NRATT_{\vec{k},t_i} \mid \vec{k} = (k_1,\ldots,k_n), k_1,\ldots,k_n \in \mathbb{N}, t_i \in T \}$.

Figure 2 depicts the attribute set relationships NRATT $\subseteq$ NATT $\subseteq$ NGATT and the node relationships RAND $\subseteq$ AND $\subseteq$ S along with the functional relationships between attributes and nodes.

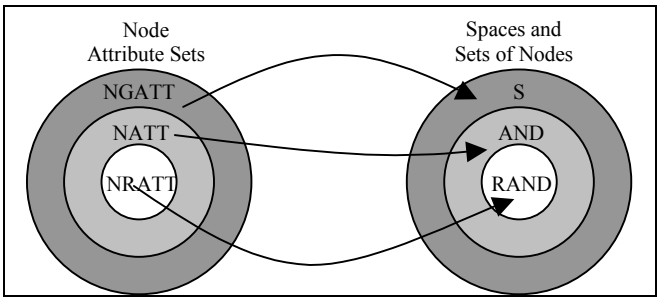

Figure 2: Concentric areas showing the set relationships between the attribute sets and nodes, with arcs showing the relationships between them.

It is possible that a given $s_{\vec{k}}$ at time t will not posses any relevant features and $\text{nratt}_{\vec{k},t}$ will be empty. The nodes corresponding to members of RATT that contain non-empty NRATT sets are depicted as the white rectangles in Figure 1, i.e. RATT is used as a constraint on the node space **AND** in order to create a set of relevant annotated nodes.

Definition 6: The set of *relevant annotated nodes* RAND $\subseteq$ AND is defined by RAND = $\{ \text{rand}_{\vec{k},t_i} \in \text{RAND},\ \text{rand}_{\vec{k},t_i} = (\text{and}_{\vec{k},t_i}, \text{nratt}_{\vec{k},t_i}) \mid \| \text{nratt}_{\vec{k},t_i} \| > 0,\ \vec{k} = (k_1,\ldots,k_n),\ k_1,\ldots,k_n \in \mathbb{N},\ \vec{l} = (l_1,\ldots,l_n),\ l_1,\ldots,l_n \in \mathbb{N},\ t_i \in T \}$.

The following definitions on graphs are based on definitions from Aho and Ullman [1]. In order to form a graph, a set of connections between nodes must be established. This connection is from a start node to an end node, and may be viewed as a binary relation between the two nodes. These binary relations, or arcs are depicted as black arrows in Figure 3.

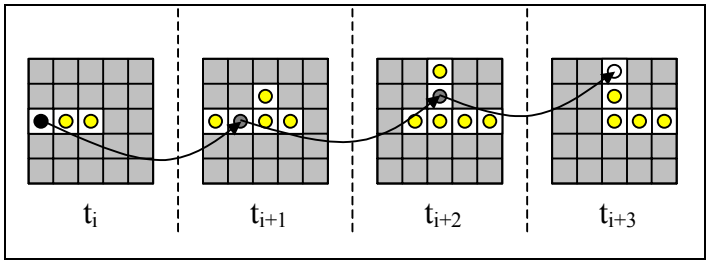

Figure 3: A path through a planning graph and the spanning set for each node in the path. The path origin is the black circle, the goal is the white circle, and the intermediate nodes are the gray circles.

Definition 7: An *arc* ∈ ARC is defined as a binary relation r on RAND. A *directed graph* consists of a set of nodes RAND and a set of arcs ARC.

Given a set of attributed nodes, a domain dependent spanning function may be defined that specifies which nodes are reachable from a given node in a single transition. These nodes are called the node's *successors* and are connected with an arc.

Definition 8: Define the *spanning set* SP of a node $rand_{\vec{k},t_i}$ as the set of all successor nodes $SP(rand_{\vec{k},t_i}) = \{sp_{\vec{k},t_{i+1}} \mid \| nratt_{\vec{k},t_{i+1}} \| > 0\}$.

The spanning set of each black node at time t is shown for the next consecutive time slice in Figure 3. The series of black nodes and the arcs that connect them form a path through the graph. The path in Figure 3 is of length 3. It should be noted that the spanning set is time dependent, and therefore the graph topology may change on a planning cycle to planning cycle basis. In addition the spanning set for node $nd_{\vec{k}}$ at time t may not be equal to the spanning set for node $nd_{\vec{k}}$ at time t+1.

Definition 9: Given a directed graph GR = (RAND, ARC), a *path* $prand(rand_{\vec{k},t_0}, rand_{\vec{l},t_m})$, $\vec{k} = (k_1,\ldots,k_n)$, $k_1,\ldots,k_n \in \mathbb{N}$, $\vec{l} = (l_1,\ldots,l_n)$, $l_1,\ldots,l_n \in \mathbb{N}$, $t_i \in T$, is defined as a list of nodes $(v_{1,t_0}, v_{2,t_1},\ldots, v_{m,t_m})$ such that $v_1 = \vec{k}$, $v_m = \vec{l}$ and there is an arc from each node to the next. The *length* m of the path is defined as the number of arcs along the path.

Definition 10: A graph is defined as an *acyclic graph* if no path begins and ends at the same node.

Lemma 1: The graph GR is an *acyclic graph*.

Proof: Assume that the path $prand(rand_{\vec{k},t_0}, rand_{\vec{l},t_0})$, $\vec{k} = \vec{l}$ exists. Then by Definition 9 there must exist an arc $r(rand_{\vec{j},t_{0+d}}, rand_{\vec{l},t_0})$ such that $d \geq 0$ and $rand_{\vec{l},t_0}$ is a successor of $rand_{\vec{j},t_{0+d}}$. However, by Definition 8, $rand_{\vec{l},t_0} \notin SP(rand_{\vec{j},t_{0+d}})$, $d \geq 0$, and $t + d \geq t$. Therefore $r(rand_{\vec{j},t_{0+d}}, rand_{\vec{l},t_0})$ does not exist and $prand(rand_{\vec{k},t_0}, rand_{\vec{l},t_0})$, $\vec{k} = \vec{l}$, cannot exist which contradicts the basic assumption.

As previously stated, one of the objectives of this system is not to find simply any path from a given start node to a given goal node, but to find a cost optimal path. Since a path

is a series of arcs, a function for computing the cost of an arc must first be defined. As stated by Fernández and González [6], this function is known as a cost function and may be defined as:

Definition 11: The mapping c:ARC→ℂ is called the cost function of an arc.

The set of costs ℂ must be at least a partially ordered set, but may also be a totally ordered set and must have the "plus" (+) operator defined. One typical specification for the set ℂ is $\mathbb{C} = \Re^{+}$.

Once the cost of individual arcs is known, it is a simple matter to compute the cost associated with an entire path.

Definition 12: The cost associated to a path prand is defined as $pc = \sum_{i=1}^{m} c(r(v_i, v_{i+1}))$

The system objective of finding the cost optimal path between two nodes may now be achieved by finding the minimum cost path.

Definition 13: Define the minimum cost path $prand_{min}$ between the nodes $rand_{\vec{k},t}$ and $rand_{\vec{l},t+m}$ where $m \neq 0$ and $\vec{k} = (k_1,\ldots,k_n)$, $k_1,\ldots,k_n \in \mathbb{N}$, $\vec{l} = (l_1,\ldots,l_n)$, $l_1,\ldots,l_n \in \mathbb{N}$ such that $pc(prand_{min}) \leq pc(prand)$, $prand \in \{q \in PRAND \mid q = prand(rand_{\vec{k},t}, rand_{\vec{l},t+m})\}$.

Integrated Incremental Graph Generation and Planning Algorithm

In a traditional graph planning approach, a graph that represents the entire discrete problem space is constructed prior to a search [2,3,11,13,15]. The problem constraints are then encoded as weights on the individual arcs, and a graph search is performed to determine the minimum cost path through the planning space. As will be shown, this has potentially unrealizable memory requirements to store the graph structure and unrealizable computational time requirements to determine graph node connections and arc costs. In addition, as stated by Russell ([14], p. 92), the only place where knowledge can be applied once the graph has been constructed is in the choice of which node to expand next.

An algorithm will now be presented that incrementally builds and evaluates a planning graph in order to find the minimum cost path $prand_{min}$ between the nodes $rand_{\overrightarrow{start},t_0}$ and

$\text{rand}_{\overrightarrow{\text{goal}},t_0+m}$. Unlike traditional graph search techniques such as breadth first, depth first, and A* [12,14], this technique provides for knowledge to be applied in both the graph construction and planning step. Only as much of the graph as is necessary to perform the task at hand will be constructed, and this construction will be interleaved with the arc cost evaluation and graph search. Through the use of this technique, both hard and soft constraints on the planning space are supported. As will be shown, this integrated incremental graph construction leads to a potentially significant reduction in the size of the graph by constraining the branching factor and the total number of nodes that must be instantiated while the arc cost function allows the soft constraints to elicit a variety of different planning behaviors from the system.

The algorithm consists of the steps illustrated in Figure 4.

1) Insert $\text{rand}_{\overrightarrow{\text{start}},t_0}$ into open list O that contains all visited nodes.
2) Insert one or more goal nodes $\text{rand}_{\overrightarrow{\text{goal}},t_0}$ into the goal list G.
3) Using the graph search technique of your choice, select and remove a node $\text{rand}_{\bar{k},t_i}$ from O.
4) If $\text{rand}_{\bar{k},t_i} \in$ G, and re-planning is desired go to (1) else if $\text{rand}_{\bar{k},t_i} \in$ G finished, else go to (5).
5) Define the set to be visited TBV = $\text{SP}_{\text{rand}\bar{k},t_i}$.
6) While TBV $\neq \varnothing$, create arc $(\text{rand}_{\bar{k},t_i}, \text{rand}_{\bar{l},t_{i+1}})$ and evaluate temporary variable $\beta = \text{pc}(\text{prand}(\text{rand}_{\overrightarrow{\text{start}},t_0}, \text{rand}_{\bar{l},t_{i+1}}))$ where $\text{rand}_{\bar{l},t_{i+1}} \in$ TBV.
   a. Remove $\text{rand}_{\bar{l},t_{i+1}}$ from the set TBV.
   b. If $\text{rand}_{\bar{l},t_{i+1}}$ marked as VISITED, and $\beta < \alpha_{\bar{l},t_{i+1}}$, set $\alpha_{\bar{l},t_{i+1}} = \beta$, set $\text{rand}_{\bar{l},t_{i+1}}$ backpointer to $\text{rand}_{\bar{k},t_i}$, and add $\text{rand}_{\bar{l},t_{i+1}}$ to O.
   c. Else if $\text{rand}_{\bar{l},t_{i+1}}$ not marked as VISITED, mark as VISITED, set $\alpha_{\bar{l},t_{i+1}} = \beta$, set $\text{rand}_{\bar{l},t_{i+1}}$ backpointer to $\text{rand}_{\bar{k},t_i}$, and add $\text{rand}_{\bar{l},t_{i+1}}$ to O.
   d. Remove arc $(\text{rand}_{\bar{k},t_i}, \text{rand}_{\bar{l},t_{i+1}})$.
   e. End while.
7) Go to 3.

Figure 4: Integrated incremental graph generation and planning algorithm.

Step 1 and 2 of the algorithm depicted in Figure 4 begin the search process by placing the node that contains the current system state into the list O of open nodes and one or more goal states into the goal set G. The algorithm will terminate when a minimum cost path has been found to any of the nodes in the set G. It should be noted that the feature of being a start or goal node is a relevant attribute and therefore the start and goal nodes are always relevant and elements of RAND. This assures that the graph will contain the start and goal nodes even if these nodes have no other relevant features associated with them.

In step 3, it is assumed that a search technique that utilizes an open list is used. Many such techniques exist in the current literature such as breadth first [14], depth first [14], and A* [8]. A search methodology dependent criterion is used to select the node from the list O that should be expanded. If the list O is empty, then a plan failure has occurred. At this point, the system may either return a failed status indicating that no plan was found, or may relax the selection criteria for the NRATT (i.e. increase the size of NRAND) and attempt to find a new plan. Through this technique, a hierarchy of hard constraints may be established that are gradually relaxed as a result of plan failures.

Step 4 performs a test to determine if the current node is a member of the goal set G. The goal set is defined as the set of nodes in RAND that will satisfy the objectives of the system. This set may contain more then one node (e.g. a mobile robot should arrive at a certain location as soon as possible, with no specific time information given), but should never be empty.

Definition 14: Define the goal set G as the set of relevant annotated nodes (RAND) such that $G = \{ g_{\bar{k},t_i} \in RAND \mid \vec{k} \in K \subseteq \mathbf{S} \wedge t_i \in T_i \subseteq T \}$.

One integral feature of this algorithm is dynamic re-planning. By returning to step 1 after a successful plan has been generated, dynamics in the hard constraints, soft constraints, and environment are easily handled. Additionally, sections of previously constructed plans are reused when possible in order to speed the re-planning process.

Steps 5 and 6 incrementally expand and evaluate the graph structure. Arcs are temporarily created to link the current node to all of its successors, and the path cost function pc evaluates the new potential path to the successor node. The least expensive path from a given node to the start of the search may be found by following a series of backpointers, and the least expensive cost from a node to the start of the search is maintained in the variable $\alpha_{\bar{l},t_i}$. Since only a single path from each node back to the start is maintained, the acyclic property of the graph is guaranteed.

Graph Generation for various search strategies

Many different search strategies exist in the literature that could be used in step (3) of Figure 4 to determine the next node to expand. These searches may be decomposed into the two broad categories of uninformed and informed search. The main difference between the two strategies is that uninformed search makes its node expansion decision based solely on the current node and nodes on the open list, while informed search takes advantage of domain-specific knowledge such as the properties of the goal node.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bi-directional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if l≥d	Yes	Yes

Table 1: Comparison of different search strategies (from Russell and Norvig [14] p. 81). In this table, b is the number of states yielded from a node expansion or the *branching factor*, d is the *depth* of the tree, l is the *depth limit* for a depth-limited search, and m is the *maximum depth* of a graph.

Uninformed Search

Table 1 ([14], p. 81) provides information on various properties about uninformed search strategies that could be used in step (3) to determine the next node to expand. Information is provided on completeness (the strategy is guaranteed to find a solution if one exists), time complexity (how long the search takes to find a solution), space complexity (how much memory is required to perform the search), and optimality (the strategy is guaranteed to find the lowest cost path). It should be noted that the memory requirement shown is the amount of memory required to maintain the list of nodes awaiting node expansion, and does not include the memory required to store the graph structure itself. The size of the graph structure is the same for all of the search strategies.

The only precondition that exists for the values given in Table 1 is on the cost function used for arc evaluation. For the breadth-first, iterative deepening, and bi-directional (assuming two breadth-first searches) searches, optimality is only guaranteed if this cost function is a non-decreasing function of the depth of a node in the graph. This condition is usually satisfied only when all arcs at a given depth have the same cost, and is not necessarily a feature of the cost of an arc c. However, the breadth-first search can be modified to become a uniform-cost search that is optimal over the cost of a path pc. For the uniform-cost search, optimality is guaranteed if the cost function is defined as $c:ARC \rightarrow \Re^+$. Since $\Re^+$ is a totally ordered set, Definition 11 allows this mapping as a valid cost function.

Uniform Cost Version of Algorithm

Uniform cost search is a modification of the standard breadth-first search that finds the cheapest goal state independent of its depth in the graph (as computed by the cost function c). In order to meet the criteria in Table 1, the only constraint on this form of search is that the cost along a path must never decrease. In other words, $\forall i \in \mathbb{N}$: $c(r(v_i, v_{i+1})) \geq 0$. Once again, the selection of $c:ARC \rightarrow \Re^+$ guarantees this condition, and **Definition 11** allows this mapping. Therefore, the only change to the generic graph generation algorithm presented above for the use of a uniform cost search algorithm is in step (3) of the algorithm. This step now becomes:

3) Select and remove a node cn from the set O of open nodes such that:
$cn \in \{ rand_{\vec{k},t_i} \mid \forall\ rand \in O: pc(prand(rand_{\overrightarrow{start},t_0}, rand_{\vec{k},t_i})) =$
$\min(pc(prand(rand_{\overrightarrow{start},t_0}, rand))\}$.

Informed Search

In addition to the search strategies mentioned in Table 1, many search strategies that take advantage of problem specific knowledge exist in the literature (e.g. Dijkstra search [4], A* [14,17], bi-directional A* [10], and D* [16]). This class of search strategies is known as informed search. The class of informed strategies that utilize parameters of the goal node is known as best-first search. This name originates from the fact that these algorithms utilize an evaluation function that returns the estimated desirability of expanding a node, and hence the "best" node is expanded first. As in the uninformed search, there are informed search strategies that are optimal and guaranteed to find a solution if one exists and those that are not. One basic best-first search technique that is guaranteed optimal and complete [14] is the A* search. A* search may be written as a function f(n) = g(n) + h(n) where g(n) is a function that computes the cost of reaching node n from the start node and h(n) is a heuristic function that estimates the cost of achieving the goal node from n. To be optimal and complete, this search requires that the function h(n) that estimates the desirability of expanding a node n never underestimates the desirability of the goal (or never overestimates the remaining cost to the goal) and a search using only g(n) is optimal and complete. The uniform cost search strategy was shown to be optimal and complete and may be used for g(n). Therefore, if h(n) is an admissible heuristic (never overestimates the cost to reach the goal) A* will be optimal and complete for the graph structure defined above.

A* Version of Algorithm

A* search is a modification of the uniform cost search presented in Figure 4. Rather then expanding the cheapest node from the open list, it expands the node whose cost plus the estimated remaining cost to the goal is cheapest. Therefore, the only change to the generic graph generation algorithm presented above for the use of an A* search algorithm is in step (3). This step now becomes:

3) Select a node cn to open from the set O of open nodes such that:
$cn = \text{rand}_{\vec{k},t_i} : \forall\, \text{rand}_{\vec{j},t_i} \in O,\ pc(prand(\text{rand}_{\overrightarrow{start},t_0}, \text{rand}_{\vec{k},t_i}) + h(\text{rand}_{\vec{k},t_i}) \leq$
$pc(prand(\text{rand}_{\overrightarrow{start},t_0}, \text{rand}_{\vec{j},t_i}) + h(\text{rand}_{\vec{j},t_i}))$, where h() is an admissible heuristic function.

Example of framework applied to Towers of Hanoi

In order to illustrate the integrated incremental graph generation and evaluation algorithm, let us examine the basic game of "Towers of Hanoi" as a simple component assembly problem. In this game, the player is given three towers, one of which contains *n* different sized stacked rings (the sub-assembly inventory). The objective of the game is to move all of the rings to the rightmost tower (the final product). The only rules are that you may only move one ring at a time, and must never allow a ring to rest upon a smaller ring. Typically, the starting configuration is to have all of the rings on the leftmost tower. However, this starting configuration may be varied.

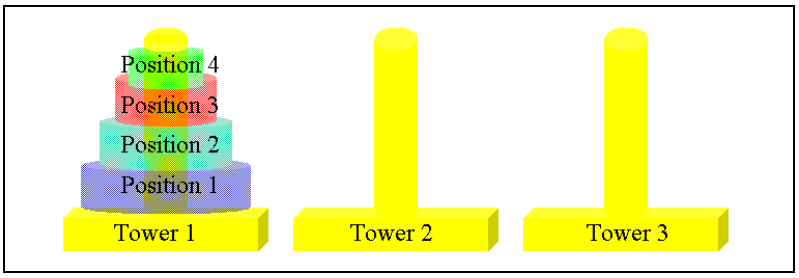

Figure 5: Illustration of Towers of Hanoi with four rings.

As shown in Figure 5, a system that contains at most *n* rings will have three towers each containing *n* ring positions. In the naïve system, where the exact number of rings (in both total quantity and number of each type) is not known, all possible combinations of rings must be mapped into nodes. The reason for this is that a minimum set of basic rules is assumed for the creation of the planning graph, and all remaining game knowledge is contained in the arc evaluation. This produces $(n+1)^n$ possible different individual tower configurations or $(n+1)^{3n}$ nodes for the entire system. Using traditional graph approaches, this will produce a graph that contains $\approx 2.4 \times 10^8$ nodes for a system of up to four rings. The branching factor for such a naïve system will be (3n-1)(2n)+n or 92. This number comes from the fact that each ring can move to 1 of 11 different positions (3n-1), each ring may be added in each of 11 positions (3n-1), and each ring may be deleted (n). As shown in Table 1, this branching factor directly impacts both the time and memory complexity of the search algorithms. Even in this simple example the problem of an unreasonable number of possible graph nodes and connections is quickly reached. Yet given this unreasonable number of nodes and connections, the system is still unable to cope with unexpected environmental changes. The reason for this is that the graph is built before the planning process begins and only anticipated actions are built into the graph.

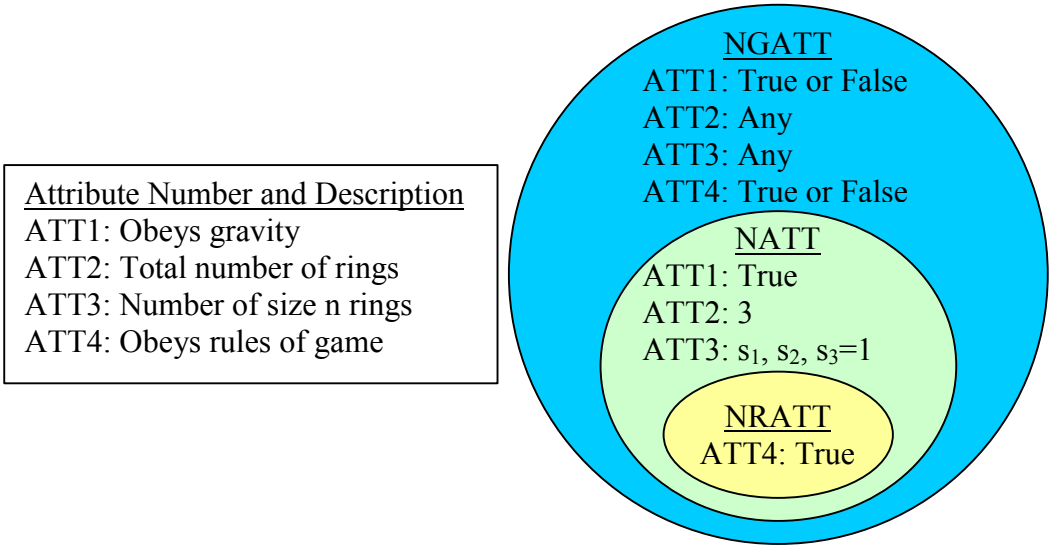

Figure 6: Description of various attribute sets.

In order to reduce the number of nodes and edges to a reasonable level, the integrated incremental graph generation and evaluation algorithm will be applied. It will be shown that this algorithm may significantly reduce the number of instantiated nodes and will reduce the branching factor to 3, regardless of the number of rings. This algorithm will also allow environmental dynamics to be represented and evaluated in the planning graph.

Before the graph generation and evaluation algorithm can be applied, the members of the attribute sets must be defined. For this particular problem, NGATT contains the four attributes of obeying gravity (spaces are not allowed between the rings on a tower), total number of rings, number of size i rings, and obeying the rules of the game (the plan is not allowed to cheat). As shown in Figure 6, NGATT contains all possible subsets of the attributes. This set corresponds to the set of nodes that would be found in a traditional planning graph. The set NATT $\subset$ NGATT restricts the node space to obey gravity and to contain one each of three different sized rings. The set NRATT $\subset$ NATT further restricts the node space to force compliance with the rules of the game.

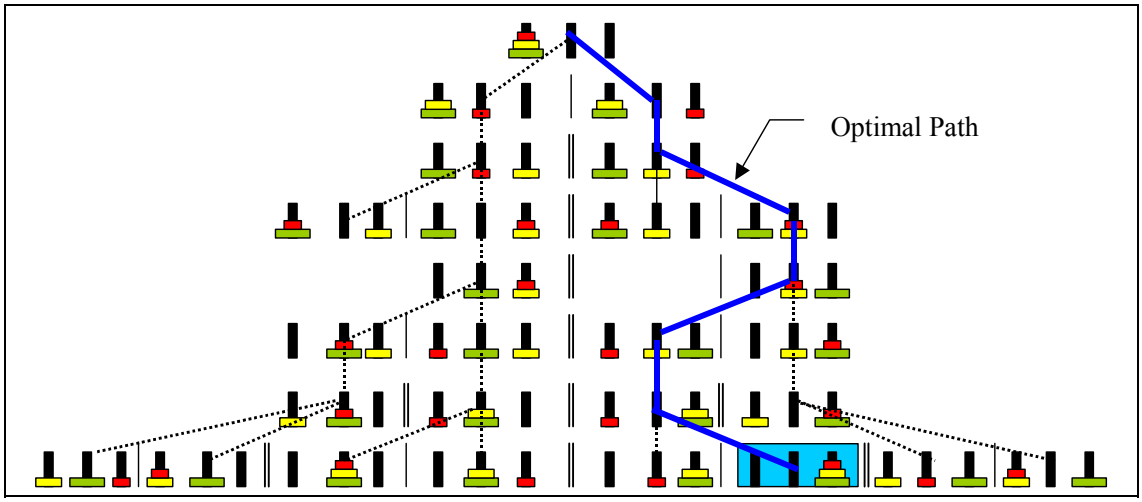

Figure 7: Optimal path found using uniform cost function and nodes RAND. In the diagram, a single line (|) separates successors of a given node and a double line (||) separates one node's successors from another's. Arcs are shown as a dotted line.

Figure 7 illustrates the optimal path found by the graph generation and evaluation algorithm. It should be noted that a traditional graph would contain 262,144 nodes. These nodes would also have to be connected by arcs. Even if there was only one arc per node (making graph search unnecessary) there would be 262,144 arcs. By using relevant attributes, the number of nodes can be reduced to only 27 nodes and 64 arcs. This low number is due to the branching factor being reduced to 3 by the constraints imposed by NRATT. In addition to this low branching factor, many of the arcs are never created due to their evaluated cost being higher or equal to an existing arc. This provides a savings of at least 4 orders of magnitude of the memory required to store the graph. In addition to the memory savings, the only nodes that need to be evaluated for arc generation are the

27 nodes that were touched by the search. In the traditional search approach, all 262,144 of the nodes would have been evaluated for arc connections before the search begins.

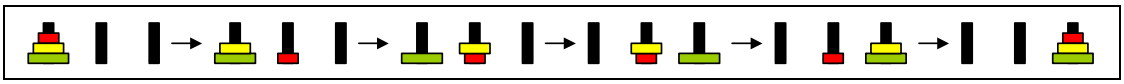

Figure 8: Towers of Hanoi solution where the rules of the game were not considered relevant.

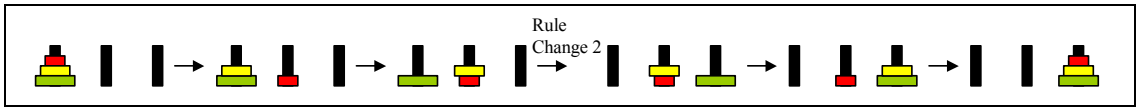

Figure 9: Towers of Hanoi solution where the rules of the game concerning which rings may be moved became relevant, but the legality of the actual tower configuration was ignored.

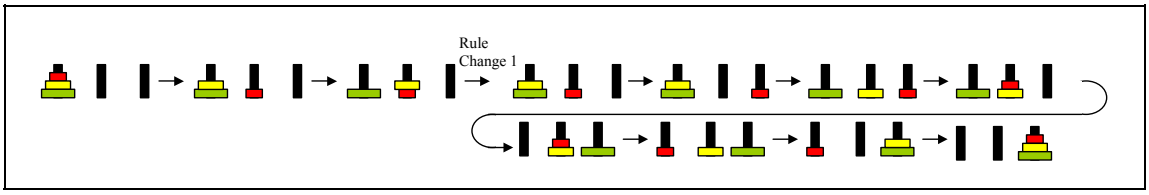

Figure 10: Towers of Hanoi solution where the rules of the game became relevant after the start of plan execution, and existing illegal tower configurations were discouraged.

As shown in Figure 8 through Figure 10, the behavior of the system may be altered through the selection of the relevant attributes. In Figure 8, NRATT = NATT and arcs representing all possible moves (both legal and illegal) are created to connect the members of NRAND as the graph is expanded. Figure 9 displays the case where the relevant rules were changed in the middle of plan execution, and a replanning cycle occured. NRATT started as NRATT = NATT and was changed after two moves were executed to the reduced set which requires ATT4 to be "true". In this case, the only arcs that are created represent legal future moves. Since the current state of the system is an illegal state (by the new NRATT definition) NRAND will contain illegal states. However, *all planned ring moves* will be legal. In contrast, NRATT in Figure 10 provides for only legal arcs to be generated and has the additional constraint implemented in the cost function that illegal states are more expensive then legal states. Upon replanning with the new rules and cost function, this causes the system to quickly undo the illegal move and put the system in a legal state before proceeding with the problem solution. It should be noted that while the system plans the optimal set of node transitions from the current time forward, dynamics might cause the global path from the initial state to be sub-optimal. As shown in Figure 10, previously performed moves must be undone thus creating a longer, sub-optimal final path.

The rule changes depicted in Figure 9 and Figure 10 occurred after the system had executed two moves. If these rule changes could have been anticipated through the use of

the time dimension of NRATT, then the plan for Figure 9 would remain unchanged, and the plan for Figure 10 would be the same as the plan from Figure 7. The reason for this is that the system would be able to anticipate the need to undo moves and therefore would not make them in the first place.

Parallel planning in partitioned problem spaces

While the set of possible general node attributes (NGATT) is the same over the entire node space **ND**, in many practical applications the possibility of certain node attributes occurring may be tied to local phenomena. It is also possible that the location of these local phenomena is tied to a subset of the possible general node attributes $\mathrm{NATT}_{R_i} \subseteq \mathrm{NATT} \subseteq \mathrm{NGATT}$. The set of relevant attributes that are selected for use in graph construction will still be of the form $\mathrm{NRATT}_{R_i} \subseteq \mathrm{NATT}_{R_i}$. A *region* may then be defined as a subset of the nodes in RAND that share certain relevant attributes.

Definition 15: A *region* R_i is defined as a subset R_i of the relevant annotated nodes RAND where $R_i = \{ \mathrm{rand}_{\vec{k},t} \mid \| \mathrm{nratt}^{R_i}_{\vec{k},t} \| > 0,\ \vec{k} = (k_1,\ldots,k_n),\ k_1,\ldots,k_n \in \mathbb{N},\ \vec{l} = (l_1,\ldots,l_n),\ l_1,\ldots,l_n \in \mathbb{N},\ t_i \in T \}$.

To complete the definition of a region, a set of regions that contains all of the nodes form RAND will be defined as a *complete* set of regions.

Definition 16: The set of n regions $R = \{R_1,\ldots,R_n\}$ is *complete* iff $\bigcup_{i=1}^{n} R_i = \mathrm{RAND}$.

It can be shown that any node, $nd \in \mathrm{RAND}$ is also a member of the complete set of regions R.

Lemma 2: If $nd \in \mathrm{RAND}$ and $R = \{R_1, \ldots, R_n\}$ is a complete set of regions, then $nd \in R$.

Proof: Assume $nd \notin R$, then by Definition 16, $R = \bigcup_{i=1}^{n} R_i = \mathrm{RAND}$ and $nd \notin \mathrm{RAND}$ which contradicts the original assumption.

One common occurrence of regions is in the decomposition of a planning space into a hierarchy. Figure 11 shows one possible technique for forming a hierarchy that was presented by Fernández and González [5,7] where high-resolution nodes are abstracted to form a smaller number of lower resolution super nodes. Each super node is then connected to form a planning graph and planning is performed on this low-resolution representation of the space. Fernández and González present a technique for connecting and searching these graph structures that is based on the classic refinement method. However, the authors state that this technique is not guaranteed to find the cost optimal path ([7], p. 106). In the refinement method, a path is first found through the low-resolution graph. In time-constrained environments, this path may be executed as a sub-optimal plan. If more time is available, individual super nodes that are part of this path

are expanded and a new refined plan is constructed that is a refinement of the previous path. Any number of hierarchical levels, referred to as a multi-hierarchy, may be imbedded in such a representation and the multi-hierarchy may act as an anytime algorithm [18].

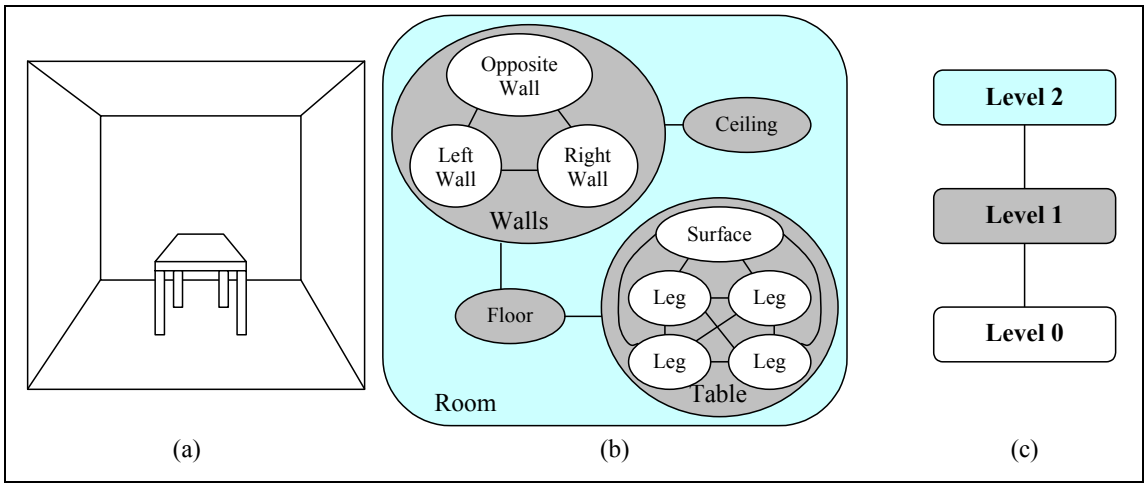

Figure 11: Example from Fernández and González ([7], p.104) of a graph-based abstraction hierarchy. (a) world view of a room, (b) hierarchical graph model with levels represented by shaded regions as keyed by (c).

Another way to view the above hierarchy [5] and multi-hierarchy [7] is through the use of regions as defined above and shown in the left hand side of Figure 14. Under this view, the set RAND would contain all of the Level 0 nodes and would be defined by the set of node relevant attributes NRATT. A super node is then defined as a region $R_i \subseteq$ RAND such that $NRATT_{R_i} \subseteq NRATT$. For planning purposes, individual nodes may then represent these regions, and a regional spanning distance may then be defined according to Definition 8 that could be used in the integrated incremental graph creation and evaluation algorithm. Modifications to the integrated incremental graph creation and evaluation algorithm that will guarantee that the cost optimal path is found are presented below.

Another view of hierarchical planning is presented by Lacaze [9] for mobile robot applications. In this paper, the planning space is viewed as having a high node density near the robot with decreasing density as the distance to the robot increases. Forming regions of equal node count as shown in Figure 12 creates the hierarchy. A region of high node density exists around the robot's current location where high-resolution sensor information is available. Further from the robot, the sensors are only able to detect large objects and the corresponding node density decreases. Finally, once the limits of the sensors have been exceeded, the node density drops to correspond to the feature density represented in the a priori dataset.

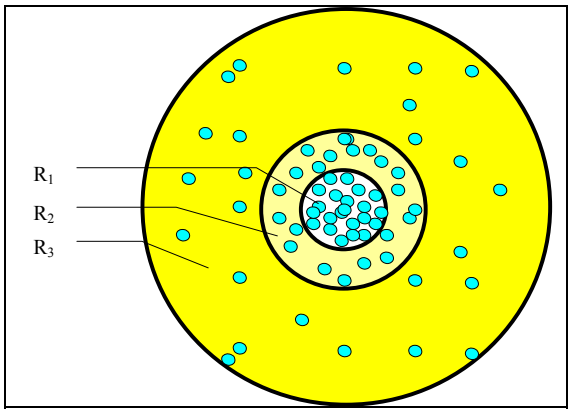

Figure 12: Example of node density for 2-D mobile robot application based on Lacaze [9]. Rings represent regions of equal node count.

The general approach of regions can once again be specialized to account for this form of hierarchy. As an example of this, examine the mobile robot application with NGATT composed of high-resolution sensed feature information and low-resolution a priori feature information. The high-resolution information's availability is tied to the local phenomena of distance from the location of the robot's sensors, while the low-resolution features are available over the entire map extents. This problem may be naturally partitioned into a region that contains high-resolution information (R_1) and a region that does not (R_2) where $NATT_{R_1} \supseteq NATT_{R_2}$. Since $NATT_{R_1} \supseteq NATT_{R_2}$, it is possible that more attributes are available for selection as members of $NRATT_{R_1}$ and that the composition of RAND in the region R_1 defined by $RATT_{R_1}$ will be different than in the region R_2 defined by $RATT_{R2}$.

Once a density pattern such as the one in Figure 12 has been detected, the node space may be naturally partitioned into regions where the predicted node count and graph search complexity will be approximately equal. Responsibility for planning in each region may then be partitioned to separate processes, and these processes communicate through the use of a graph construction technique such as the one presented by Lacaze [9].

Lacaze [9] describes this technique for use in hierarchical planning where regions of decreasing node density exist in concentric areas that originate at the system's current location (much as the concentric circles depicted in Figure 12) and the levels of the hierarchy plan at different rates. The only additional precondition for the use of this technique is that the reverse arc cost function c′ must exist.

Definition 17: Define the reverse arc cost function c′ as the mapping $c': ARC \rightarrow \Re^+ \mid c(arc(rand_{\vec{k},t}, rand_{\vec{l},t+d})) = c'(arc(rand_{\vec{l},t+d}, rand_{\vec{k},t}))$, $\vec{k} = (k_1,\ldots,k_n)$, $k_1,\ldots,k_n \in \mathbb{N}$,

$\vec{l} = (l_1,\ldots,l_n)$, $l_1,\ldots,l_n \in \mathbb{N}$, $d \in \mathbb{N}$, $t_i \in T$. Also, define the reverse cost associated to a path prand as $pc' = \sum_{i=1}^{m} c'(r(v_i, v_{i+1}))$.

This cost function is necessary in the algorithm implemented by Lacaze [9] because he plans from the goal node to the start node. In the following sections, Lacaze's technique [9] will be formalized, and it will be shown that a generalization of this technique may be viewed as a specific implementation of the planning framework described in this paper. Lacaze's technique will then be extended to apply to the more general case of an arbitrary number of regions with arbitrary boundaries.

The algorithm described below plans from the start node to the goal node, however goal to start planning may be performed by exchanging the start and goal nodes and using the cost function pc′. The path obtained through this exchange will have the same minimum cost, but may contain different nodes. The reason for this is that the search algorithm is complete and optimal and will give a path with a cost equivalent to the minimum cost path from start to goal. However, if more then one equivalent minimum cost path exists from start to goal, different paths may be found by the two approaches.

During the operation of the planning algorithm that is described in Figure 13, it is important to know if a node is on the boundary or in the interior of a given region. A node will be defined to be on the boundary of regions R_a and R_b if it is a member of both regions and its spanning set includes at least one node that is an exclusive member of each region.

Definition 18: Given the set of n regions $R_1,\ldots,R_n$, the *boundary set* of any two regions i,j is defined as $B_{i,j} = \{\ rand_{\bar{k},t} \in R_i, R_j \mid \exists\ \text{node}\ rand_{\bar{l},t} \in SP(rand_{\bar{k},t})$ such that $rand_{\bar{l},t} \notin R_i$ and $rand_{\bar{l},t} \in R_j \wedge \exists\ \text{node}\ rand_{\bar{l},t} \in SP(rand_{\bar{k},t})$ such that $rand_{\bar{l},t} \in R_i$ and $rand_{\bar{l},t} \notin R_j\ \}$.

Two regions R_i, R_j will be denoted as shared boundary regions if their intersection is equal to the boundary region between the two sets. Shared boundary regions have "shallow" boundary regions that minimize the region overlap.

Definition 19: Two regions R_i, R_j are called *shared boundary regions* SBR_i and SBR_j iff $R_i \cap R_j = B_{i,j} = B_{j,i}$.

In order to specialize the integrated incremental graph generation and planning algorithm from Figure 4, a new notion of a spanning set that is specialized for regions must be defined. It will be shown that the union of these regional spanning sets is equal to the spanning set defined in Definition 8.

Definition 20: Define the *regional spanning set* SPR_i of a node $rand_{\bar{k},t_i}$ as the set of all successor nodes $SPR_i(rand_{\bar{k},t}) = \{sp_{\bar{k},t+1} \mid ||nratt_{\bar{k},t+1}|| > 0 \wedge sp_{\bar{k},t+1} \in R_i\}$.

Lemma 3: $\forall\ rand_{\bar{k},t} \in RAND$, $SP(rand_{\bar{k},t}) = \bigcup_i SPR_i(rand_{\bar{k},t})$ where $R = \{R_1, \ldots, R_n\}$ is a complete region.

Proof:

Definition **20** may be rewritten as $\bigcup_i SPR_i(rand_{\bar{k},t}) = \{sp_{\bar{k},t+1} \mid ||nratt_{\bar{k},t+1}|| > 0 \wedge sp_{\bar{k},t+1} \in \bigcup_i R_i\}$. From Definition 16 and the fact that R is complete, $\bigcup_i SPR_i(rand_{\bar{k},t}) = \{sp_{\bar{k},t+1} \mid ||nratt_{\bar{k},t+1}|| > 0 \wedge sp_{\bar{k},t+1} \in RAND\}$. By Definition 6, the condition $sp_{\bar{k},t+1} \in RAND$ is redundant, and the definition reduces to $\bigcup_i SPR_i(rand_{\bar{k},t}) = \{sp_{\bar{k},t+1} \mid ||nratt_{\bar{k},t+1}|| > 0\} = SP(rand_{\bar{k},t})$ by Definition 8.

Planning algorithm for SBR_r where $R=\{R_1, \ldots, R_n\}$. All nodes start as not VISITED.

1) If region contains $rand_{\overline{start},t_0}$, insert this into region's open list O_r that contains all VISITED nodes.
2) If region contains one or more goals, $rand_{\overline{goal},t_0}$, insert them into G_r.
   a) Form the set $G^* = G_r \cup B(r,j)$, $j \in \{1, \ldots, n\}$.
3) Using the graph search technique of your choice, select and remove a node $rand_{\bar{k},t_i}$ from the O_r.
4) If $rand_{\bar{k},t_i} \in G_r$, go to (2).
   a) Else, if $rand_{\bar{k},t_i} \in G^*$, remove $rand_{\bar{k},t_i}$ from G^*. If $G^* = \varnothing$, go to (2).
5) Define the set to be visited, $TBV_r = SP_{rand\bar{k},t_i}$.
6) While $TBV_r \neq \varnothing$, create arc $(rand_{\bar{k},t_i}, rand_{\bar{l},t_{i+2}})$ and evaluate temporary variable $\beta = pc(prand(rand_{\overline{start},t_0}, rand_{\bar{l},t_{i+1}}))$ where $rand_{\bar{l},t_{i+1}} \in TBV_r$.
   a. Remove $rand_{\bar{l},t_{i+1}}$ from the set TBV_r.
   b. If $rand_{\bar{l},t_{i+1}}$ marked as VISITED, and $\beta < \alpha_{\bar{l},t_{i+1}}$, set $\alpha_{\bar{l},t_{i+1}} = \beta$, set $rand_{\bar{l},t_i+1}$ backpointer to $rand_{\bar{k},t_i}$, and add $rand_{\bar{l},t_{i+1}}$ to O_r. If $rand_{\bar{k},t_i} \in B(r,j)$, $r \neq j$, $j \in \{1, \ldots, n\}$, add $rand_{\bar{l},t_i+1}$ to O_j.
   c. Else if $rand_{\bar{l},t_i+1}$ not marked as VISITED, mark as VISITED, set $\alpha_{\bar{l},t_{i+1}} = \beta$, set $rand_{\bar{l},t_{i+1}}$ backpointer to $rand_{\bar{k},t_i}$, and add $rand_{\bar{l},t_{i+1}}$ to O_r.
   d. Remove arc $(rand_{\bar{k},t_i}, rand_{\bar{l},t_{i+1}})$
   e. End while.
7) Go to 3.

Figure 13: Integrated incremental graph generation and planning algorithm specialized for planning in partitioned spaces. Each SBR executes its own version of the algorithm.

Definition 21: Define a *complete cycle* as the execution of step 6 of the algorithm of Figure 13 without changing any α values.

Figure 13 describes a specialization of the integrated incremental graph generation and planning algorithm from Figure 4 for operation over a partitioned planning space. A separate version of this algorithm will be executed for each of the partitions $R_i \in R$, $i = \{1, \ldots, n\}$. These algorithms may be run serially, or may be executed in parallel with no effect on the final complete solution as will be shown in Theorem 1. The only change to the basic framework that is necessary for operation over partitions concerns the inclusion of the boundary nodes into the algorithm framework. These boundary nodes are made part of the goal set for the partition, and must be reached in regions that do not include the goal before the algorithm can terminate.

In order to include the boundary as part of the goal set, step (2) forms the new goal set G* that includes the systems goals as well as the boundary regions. One of the main features of the algorithm is that the path cost from a boundary node $nd \in B(R_i, R_j)$ to the start node may be changed by the search algorithm running in either region R_i or R_j. In order to properly propagate this value to the region not responsible for the value change, the stopping criterion in step (4) must be changed to cause repeated planning cycles to occur. The repeated cycle will quickly terminate (no nodes other then the boundary nodes will be placed on the open list) if no changes have occurred. However, if boundary changes have occurred, they will propagate through the graph.

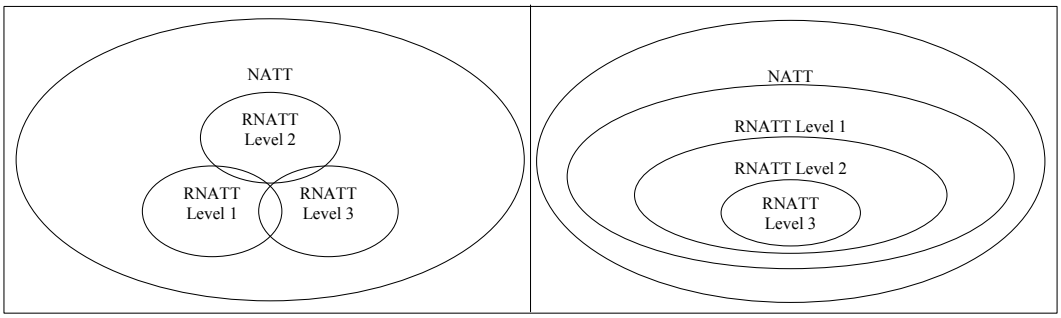

Figure 14: Set relationships for partitioned planning space.

The algorithm from Figure 13 may be applied to any configuration of shared boundary regions. Two possible configurations are shown in Figure 14, where the left hand drawing represents a hierarchy of the form of Fernández and González [7], and the right hand drawing represents a hierarchy of the form of Lacaze [9]. In general, these partitions may exist anywhere in the planning space with a separate graph search system operating on each partition. If the individual plans constructed in each partition are optimal, then the final plan constructed by the algorithm of Figure 13 will also be optimal.

Theorem 1: Given the boundary regions $R = \{R_1, \ldots, R_n\}$ with a independent version of the planning algorithm from Figure 13 executing in each region using an algorithm that is

complete on a standard planning space for node selection. Further, given the start node $rand_{\bar{s},t_i}$, and goal node $rand_{\bar{g},t_i+m}$ the path $prand(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m})$ is produced once all algorithms have finished a complete cycle. Then $prand(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m}) = prand_{min}(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m})$ is the cost optimal path through the graph in the regions R, and it is the same path as the cost optimal path through the graph in RAND.

Proof (a): Assume $prand(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m}) \neq prand_{min}(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m})$.
Then $\exists\ prand_{min*}(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m}) : pc(prand_{min*}(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m})) < pc(prand_{min}(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m}))$.
By Definition 9 $\exists$ node nd : $nd \in R$, $nd \notin prand_{min}(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m}) \wedge pc(prand_{min*}(rand_{\bar{s},t_i}, nd)) + pc(prand_{min*}(nd, rand_{\bar{g},t_i+m})) < pc(prand_{min}(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m}))$ where nd is the first node from the list V: $v_i \in V(prand_{min*}) \wedge v_i \notin prand_{min}(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m})$.
Without loss of generality, assume that $v_{i-1} \in R_i$.
Four different situations exist with respect to boundary set membership for the nodes nd and v_{i-1}.
Case 1: $v_{i-1} \notin B(R_i, R_j) \wedge nd \notin B(R_i, R_j)$.
Case 2: $v_{i-1} \notin B(R_i, R_j) \wedge nd \in B(R_i, R_j)$.
Case 3: $v_{i-1} \in B(R_i, R_j) \wedge nd \notin B(R_i, R_j)$.
Case 4: $v_{i-1} \in B(R_i, R_j) \wedge nd \in B(R_i, R_j)$.

For case 1 and case 4, both v_{i-1}, $nd \in R_i$, $i \in \{1, \ldots, n\}$. In these cases, both nodes are in the same region and a cost optimal algorithm is running in that region. The fact that nd is not included in the final path contradicts the assumption that the individual region's algorithm is cost optimal.

For case 2, $nd \in R_j$, $nd \notin SP_{R_i}(v_{i-1})$, $i \neq j \in \{1, \ldots, n\}$. By Definition 17, $\exists$ arc(v_{i-1}, nd), $nd \in SP_{R_j}(v_{i-1})$.
Therefore either nd is on open list O_j, or nd has already been expanded. Since given that algorithm cycle complete, all nd | $pc(prand(rand_{\bar{s},t_i}, nd)) < pc(prand(rand_{\bar{s},t_i}, rand_{\bar{g},t_i+m})) \notin O_j$. This forces the conclusion that nd has been expanded. However, if nd has been expanded, then by step 6 of algorithm, $prand_{min}$ would include this node. This contradicts the initial assumption.

For case 3, $v_{i-1} \in R_i$, $nd \in R_i, R_j$, $nd \in SP_{R_i}(v_{i-1})$, $i \neq j \in \{1, \ldots, n\}$. By Definition 17, $\exists$ arc(v_{i-1}, nd), $nd \in SP_{R_i}(v_{i-1})$.
Therefore nd on open list or expanded for region R_i. If nd is expanded in region R_i, then by step 6 of algorithm nd will either be on O_j, or $prand_{min}$ would include this node. $prand_{min}$ including this node contradicts the initial assumption. Therefore $nd \in O_i \vee O_j$.

However, since given that algorithm cycle complete, all nd | pc(prand($rand_{\bar{s},t_i}$, nd)) < pc(prand($rand_{\bar{s},t_i}$, $rand_{\bar{g},t_i+m}$))$\notin O_i$, O_j. This forces the conclusion that nd has been expanded in both regions which contradicts nd $\in O_i \vee O_j$.

Proof (b): From Lemma 2 and
Lemma **3**, every node and arc that exist in G_R also exists in G_{RAND}.
Therefore all paths that exist in R exist in RAND, and all paths that exist in RAND exist in R.

As previously detailed, one common form of shared boundary region is a hierarchy. The hierarchy from the mobile robot example is covered by the general algorithm presented above by constructing the complete set of regions R such that R_1 and R_n have only one boundary region (with R_2 and R_{n-1} respectively) and all intermediate regions i have two regions (one with R_{i-1} and one with R_{i+1}). A three level hierarchy of this form is shown on the right hand side of Figure 14.

Example of framework applied to mobile robot

Differences in the availability of environmental data may not be the only difference between the $RATT_i$ subsets for the different levels of a planning hierarchy. Differences in planning objectives for the levels may dictate that other constraints are to be placed on the selection of RATT. For example, the highest resolution level of planning may be concerned with creating the optimal dynamically correct path while a lower resolution level is concerned with computing the optimal set of roads to traverse to reach the objective. The fact that a path must be dynamically feasible to be considered is defined as a hard constraint that is formally defined by Definition 22. These hard constraints may be seen as not only controlling which attributes are possible for nodes to contain, but also as shaping the topology of the graph by controlling which nodes, and therefore which node transitions are possible.

Definition 22: A *hard constraint* is defined as an element of NRATT where each attribute is defined by a predicate.

In addition to the constraints that are placed on the selection criteria for the inclusion of attributes into NRATT, the inclusion of nodes into a path that contain certain attributes may be more desirable than the inclusion of nodes that do not contain these attributes. This notion of desirability may be formalized by the notion of soft constraints. Soft constraints are implemented through the arc cost function and control a cost ranking that signifies which arcs are more desirable.

Definition 23: A *soft constraint* is defined as an ordering placed on the arcs by the cost function c.

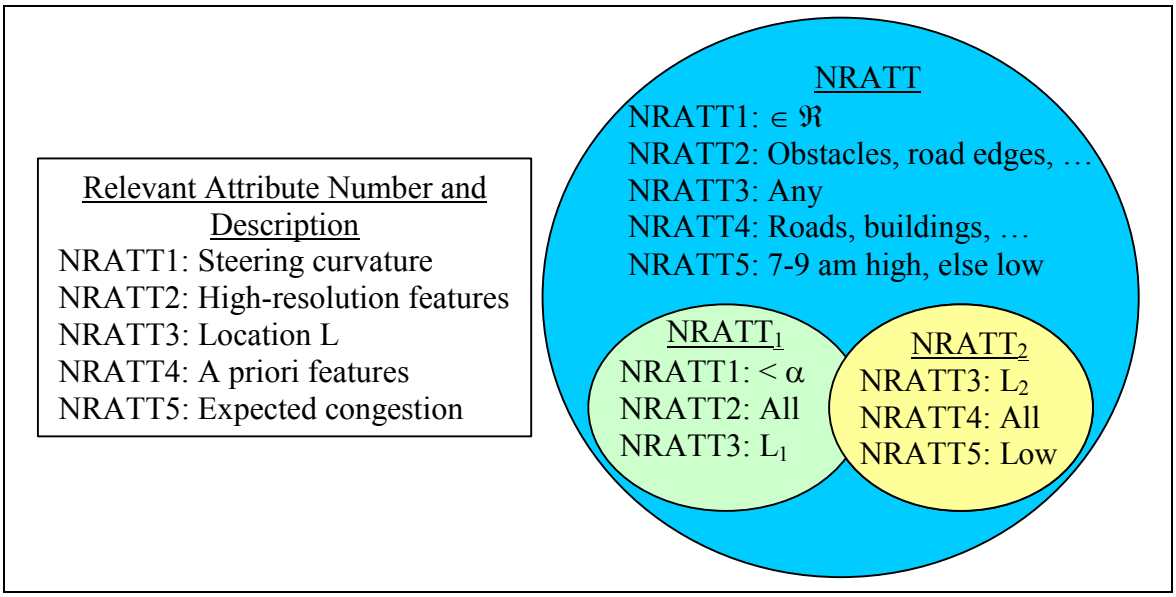

Figure 15: Example of attribute subsets used to generate regions for a hierarchical mobile robot application.

Constraint Set 1		Constraint Set 2	
Constraint	Cost/Meter	Constraint	Cost/Meter
NRATT4 = Highway ∧ NRATT5 ≠ Low	50.0	NRATT4 = (Highway ∨ Minor Road) ∧ NRATT5 ≠ Low	100.0
NRATT4 = Minor Road	1.0	NRATT4 = Off road	2.0
NRATT4 = Highway ∧ NRATT5 = Low	1.5	NRATT4 = (Highway ∨ Minor Road) ∧ NRATT5 = Low	1.0
NRATT4 ≠ Highway ∨ Minor Road	200.0		

Figure 16: Possible soft constraint sets based on NRATT from Figure 15 for region R_2.

Examples of hard constraints may be seen in Figure 15 while two different sets of soft constraints for the cost function are shown in Figure 16. The hard constraints for region R_2 will assure that no path is ever considered that contains areas of high traffic congestion. The soft constraints for constraint set 1 favor travel on minor roads (even if the path is longer) and severely penalize path segments that lie off-road. For constraint set 2, off-road is twice as expensive as on-road travel, while travel on congested roadways is penalized. This combination of hard constraints through planning regions coupled with soft constraints through the arc cost function provides the integrated incremental graph creation and evaluation algorithm a large degree of flexibility in how plans may be constructed to satisfy the system's objectives.

Acknowledgement

The authors would like to acknowledge Alberto Lacaze for his help and discussions in regard to extending the integrated incremental graph construction and evaluation algorithm to multi-level hierarchies.

References

1. Aho, A. V. and Ullman, J. D., *Foundations of Computer Science*, C Edition ed., Computer Science Press, New York, 1995.

2. Choset, H., Konukseven, I., and Burdick, J., "Mobile Robot Navigation: Issues in Imlementating the Generalized Voronoi Graph in the Plane," *Proceedings IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 1996, pp. 241-248.

3. Crowley, J. L., "Navigation for an Intelligent Mobile Robot," *IEEE Journal of Robotics and Automation*, Vol. RA-1, No. 1, 1985, pp. 31-41.

4. Dijkstra, E. W., "A note on two problems in connexion with graphs," *Numerische Mathematik*, Vol. 1, 1959, pp. 269-271.

5. Fernández-Madrigal, J.-A. and González, J., "Hierarchical Graph Search For Mobile Robot Path Planning," *Proceedings of the 1998 International Conference on Robotics and Automation*, Vol. 1, 1998, pp. 656-661.

6. Fernández-Madrigal, J.-A. and González, J., *Multi-Hierarchical Representation of Large-Scale Space*, Vol. 24, Kluwer Academic Publishers, Boston, 2001.

7. Fernández-Madrigal, J.-A. and González, J., "Multihierarchical Graph Search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, No. 1, 2002, pp. 103-113.

8. Hart, P. E., Nilsson, N. J., and Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, 1968, pp. 100-107.

9. Lacaze, A., "Hierarchical Planning Algorithms," *SPIE 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls*, 2002.

10. Manzini, G., "BIDA*: An Improved Perimeter Search Algorithm," *Artificial Intelligence*, Vol. 75, No. 2, 1995, pp. 347-360.

11. Nagatani, K., Iwai, Y., and Tanaka, Y., "Sensor Based Navigation for Car-Like Mobile Robots Using Generalized Voronoi Graph," *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 2, 2001, pp. 1017-1022.

12. Nilsson, N. J., *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann Publishers, Inc., San Francisco, 1998.

13. Nissoux, C., Siméon, T., and Laumond, J.-P., "Visibility Based Probabilistic Roadmaps," *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vol. 3, 1999, pp. 1316-1321.

14. Russell, S. and Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice-Hall 1995.

15. Saab, Y. and VanPutte, M., "Shortest Path Planning on Topographical Maps," *IEEE Transactions on Systems Man and Cybernetics*, Vol. 29, No. 1, 1999, pp. 139-150.

16. Stentz, A., "Optimal and Efficient Path Planning For Unknown and Dynamic Environments," *International Journal of Robotics and Automation*, Vol. 10, No. 3, 1995, pp. 89-100.

17. Winston, P. H., *Artificial Intelligence*, Second ed., Addison-Wesley Publishing Company, Reding, Massachusetts, 2001.

18. Zilberstein, S., "Using Anytime Algorithms in Intelligent Systems," *AI Magazine*, Vol. 17, No. 3, 2002, pp. 73-83.